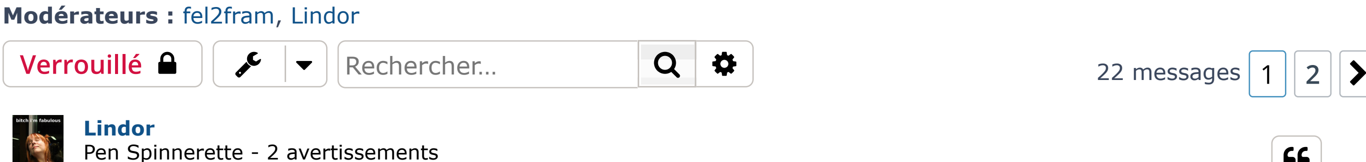


[NOTATIONS] Un langage informatique descriptif des combos

Modérateurs : fel2fram, Lindor



[NOTATIONS] Un langage informatique descriptif des combos

lun. 12 déc. 2011 18:13

Je pensais mettre ça dans le topic de Skatox, mais finalement mon objectif n'est pas le même que le sien ; nos "travaux" sont donc à dissocier, à mon sens.

Notation formelle du pen spinning

Objectif :

Définir un système de notation permettant de décrire de façon non-ambiguë tout enchaînement de figures de pen spinning. Cette notation doit rendre aisée la description d'une figure à une personne donnée, et elle doit pouvoir de plus être évolutive, c'est-à-dire être aisément modifiable et améliorable.

Forme :

J'ai choisi d'écrire cette notation sous la forme d'un langage de programmation. Deux raisons à cela : un langage de programmation est par essence même non-ambigu, en ce qu'il peut être interprété par une machine. De plus, cette forme permet théoriquement la création d'un interprétateur de ce langage, sous la forme d'un logiciel qui prendrait un programme de type « combo » en entrée et donnerait en sortie, soit un descriptif « lisible » du combo, soit une vidéo montrant le combo - ladite vidéo n'ayant pas pour but de ressembler à une vidéo réelle, mais plutôt à une suite saccadée de mouvements aisés à distinguer et à reproduire qui, mis fluidement bout à bout par l'utilisateur, redonneraient le combo complet.

NB : ce langage n'a en aucun cas pour but de mettre en évidence la manière dont on doit exécuter les tricks. Seule la trajectoire du mod et les mouvements de main faisant partie du trick sont décrits ; les moyens de réaliser concrètement le trick, en tenant compte de la gravité par exemple, ne sont pas pris en compte. Typiquement, dans la description formelle d'un bust, le mouvement de poignet est négligé.

Qu'est ce que l'orienté objet ?

La programmation orienté objet est un type de langage de programmation. Elle s'oppose par exemple à la programmation impérative (le C,…) dans laquelle un programme principal fait appel à des sous-programmes définissant des fonctions à utiliser. La programmation orientée objet fonctionne d'une manière bien plus adaptée au PS : on créé des objets qui communiquent entre eux et interagissent par envoi de message. L'idée est donc très simple : utiliser ce type de programmation et créer des objets comme « la main » et « le mod » qui interagiront. Ce système permet donc de modéliser informatiquement un combo.

Je me suis fortement inspiré du langage JAVA, simplement parce que je le connais, mais ce que j'ai conçu n'est bien évidemment pas du JAVA : ce dernier n'est pas adapté à la notation d'un combo, de toute évidence.

Détaillons un peu plus l'orienté objet : des répertoires, appelés « package » contiennent des classes. Qu'est-ce

qu'une classe ? C'est là que ça devient un peu plus technique : une classe est une « idée d'objet ». Je pense qu'un exemple peut aider à comprendre : si je créé la classe « main », j'aurais créé d'une certaine façon l'idée de l'objet main. Cette classe décrit ce qu'est une main : c'est un objet possédant une paume, un dos, 5 doigts, ces derniers pouvant saisir des objets…

Comment utiliser cette classe ? En l'instanciant. Je dispose du « schéma général de la main », sous la forme de la classe « main », et je vais donc créer un nouvel objet de type main : je déclare que cet objet est une main (« main » étant donc défini par la classe associée), et j'entre les caractéristiques de mon objet précis (par exemple, je pourrais préciser si c'est une main droite ou gauche, la longueur des doigts, etc…). Un objet, donc, c'est une classe qui a été instanciée.

Que contient une classe ? Une classe contient deux choses, qui suffisent à décrire tous les objets qu'elle définit :

-Des attributs, qui sont en quelque sorte ses caractéristiques
-Des méthodes, qui définissent ce que la classe peut faire. On peut aussi associer une méthode à un package.
Dans la classe, les attributs sont des variables : on leur affectera une valeur précise au moment d'instancier la classe, c'est-à-dire de créer un objet à partir de cette classe.

<u>Exemple :</u>

Classe « mod » :
Attributs : longueur, single sided ou double sided, longueur du corps, position du COG, largeur du corps, largeur des caps, longueur de la partie grippée, poids
Méthode : tourner ; cette méthode prend en entrée un angle et un sens de rotation

Maintenant, je vais créer un objet de type « mod » qui correspondra à un rsvp mx :
Rsvp_mx = new mod(18cm, single sided, 12cm, COP+0.5cm, 1cm, 1.1cm, 3cm, 14g) (données mises un peu au pif, bien sûr).
Et si je veux faire tourner mon rsvp mx de 180° dans le sens des aiguilles d'une montre :
Rsvp_mx.tourner(180°, CW)

Voilà pour le principe de l'orienté objet appliqué au pen spinning. Je vais maintenant définir précisément les packages, classes, méthodes, attributs et opérateurs à partir desquels j'estime pouvoir noter à peu près n'importe quel combo.

Packages et classes :

Je choisis de disposer de deux packages : h (pour « hand ») et m (pour « mod »). Oui, ici h et m sont des répertoires, et non plus des classes. Après réflexion, ça m'a paru plus pratique ainsi.

Package h :
Class Palm(direction)
Class Fingeri(position) => ici, remplacer I successivement par 1, 2, 3, 4 ou T.
Class BackHand(direction)

Mon répertoire h contient donc 7 classes : la paume, qui prend en entrée une direction (up, down, right side, left side) ; les 5 doigt, qui prennent en entrée une position (folded ou unfolded) ; le dos de la main, qui prend en entrée une direction.
Au moment de créer un objet, il est possible d'en préciser les attributs, mais ce n'est pas obligatoire.

<u>Exemples :</u>

Paume1 = new Palm() : Je créé un objet de type paume, appelé Paume1, sans donner d'indications sur sa direction (si elle est sans importance, pourquoi l'indiquer ?)

Paume2 = new Palm(up) : Je créé cette fois un objet de type paume, appelé Paume2, qui est positionné vers le haut. Je pourrais par exemple utiliser cet objet pour définir le palmspin.

Package m :
Class Side1(cap ou tip, longueur, épaisseur)
Class Side2(cap ou tip, longueur, épaisseur)
Class Body(longueur, épaisseur)
Method turn(angle, CW ou CCW, dimension)

Mon répertoire m contient donc un corps et deux extrémités. Les extrémités peuvent être créées en précisant ou non si elles sont de type « cap » ou « tip », et en donnant leur longueur et leur épaisseur.
De plus, j'associe au package m une méthode « turn » qui porte donc sur l'ensemble des classes contenues dans le package. Cette méthode prend en entrée un angle, un sens de rotation et une dimension (cad 1D, 2D ou 3D. Exemples : fingerroll = rotation en 1D ; palmspin = rotation en 2D ; charge = rotation en 3D).

Opérateurs :

Pour matérialiser les interactions entre les classes ou entre les objets, je distingue plusieurs types d'opérateurs :

Opérateurs physiques :

- || = « Collé à »
- / = « Sur »
- | = « en contact avec »
- X = « croisé avec »
-{ ; } = « suivi de »

Ces opérateurs portent sur des objets instanciés. Pour les définir précisément, il faudrait les présenter comme des fonctions qui à deux objets associent un objet ; qui plus est, il faudrait préciser l'ensemble de départ : les opérateurs ne peuvent pas nécessairement porter sur tous les objets (typiquement, Palm||BackHand n'a aucun sens…). Mais je laisse pour l'instant la description simplifiée. Ils transforment donc deux objets en un seul objet :

Finger1||Finger2 est l'objet constitué de l'index et du majeur collés.

NB : par soucis de simplification, je poserai désormais, pour i vallant 1, 2, 3, 4 ou T, Fingeri = i.

Opérateurs abstraits :

- V = « union »
- \ = « privé de »
- Wh = « whole ». C'est un raccourcis : h.Wh est la classe constitué de la réunion de toutes les classes du package h - c'est-à-dire la main entière.

Je définis l'opérateur unaire « ! » : il peut agir sur les opérateurs physiques ou abstraits, ou encore sur des méthodes. Exemple : !| signifie « jamais en contact avec » ; !turn signifie « reste immobile ».

Enfin, je définis les opérateurs de méthode : ce sont des opérateurs liés à une méthode donnée qui ne s'appliquent que lorsque la méthode est utilisée. Par exemple, l'utilisation de la méthode turn introduit l'opérateur « autour », noté « o », opérateur unaire prenant un objet quelconque en entrée.

Noter l'instanciation

Comme vous l'avez vu, rien ne distingue à première vue une classe d'un objet qui serait une instance de cette classe pour laquelle aucun attribut n'a été précisé. Par exemple, l'objet « Palm », sans indication sur sa direction,

n'est pas visiblement différent de la classe « Palm ». Ce n'est absolument pas gênant pour un utilisateur, bien au contraire ; en revanche, informatiquement, la distinction est indispensable. Si un programme devait pouvoir lire les notations, il faudrait marquer cette différence. On le fait ainsi :
Lorsqu'une classe est instanciée, on la souligne.

Exemples :

1V3 est l'objet formé par l'index et l'annulaire. C'est donc l'objet que constitue l'ensemble de ces deux doigts, sans précisions sur leurs positions respectives : abstraitement, ils sont « réunis », c'est-à-dire « considérés ensembles », et cette réunion est ensuite instanciée.
1|3 est l'objet formé par l'index et l'annulaire en contact. Les deux objets sont d'abord instanciés, puis ils sont ensuite « fondus » en un nouvel objet, constitué de ces deux objets en contact. Leur liaison est désormais physique, et l'instanciation intervient nécessairement avant l'application de l'opérateur.
La notation a été définie par soucis de rigueur, mais je ne l'utiliserai pas par la suite, ce texte ayant vocation d'être lu par un humain et non par un ordinateur.

Enfin, un dernier petit paragraphe de définitions avant de passer à l'utilisation « en pratique » :

Objet statique ou objet dynamique

Lorsque j'instancie une classe, je créé un objet. Lorsque j'applique un opérateur entre deux classes et que je les instancie ensuite, je créé un objet. Lorsque je relie deux objets par un opérateur, je créé un nouvel objet. Mais que fais-je lorsque j'applique une méthode à un objet ?

C'est là qu'interviennent les notions d'objet statique et d'objet dynamique. Par définition, un objet dynamique est un objet auquel a été appliquée une méthode : c'est donc une succession continue d'objets statiques. Cependant, si la distinction existe, les deux sont des objets et seront utilisés comme des objets : les opérateurs physiques s'appliquent aussi bien sur les objets dynamiques que statiques.
Une précision important tout de même, avant de rendre tout cela plus clair par un exemple : Un objet dynamique est à l'origine un objet statique, auquel on a appliqué une méthode. Mais ceci ne donne aucune indications sur la position finale : un objet statique (une position de la main et du mods, généralement) est défini ; une méthode lui est appliquée (turn, le plus souvent), et l'objet devient un objet dynamique, mais rien n'indique la position finale : si l'on souhaite (ce n'est pas obligatoire !) la préciser, il faudra donner, après l'objet dynamique, l'objet statique correspondant à la nouvelle position.

Exemple et première utilisation concrète de la notation :

Je souhaite définir l'objet dynamique « ThumbAround ».

Je reviens rapidement sur un opérateur physique listé plus tôt : « { ; } ». Cet opérateur défini la séquence : il clôt l'énoncé d'un objet, et permet de donner l'objet suivant. On l'utilise ainsi :
{Objet1 ; Objet2 ; Objet3} est l'objet constitué de la séquence Objet1 puis Objet2 puis Objet3.

Je rappelle que la méthode turn introduit un opérateur « o » qui signifie « autour » et qui prend en entrée un objet quelconque (on note : o(objet)). Il s'utilise uniquement après la méthode turn.

Nous allons donc créer l'objet ThumbAround_T2_T1. « ThumbAround_T2_T1 » sera son nom ; c'est une variable à laquelle on associe (via le symbole d'affectation « = ») un objet que je vais définir :

ThumbAround_T2_T1 = {
T/m.Wh.turn(2Pi, CCW, 2D)o(T)/1V2 ;
T/m.Wh/1 ; }

Notes : comme prévu, je n'ai pas souligné les objets instanciés. Mais il faudrait le faire pour être formel.
Le premier des deux objets est donc l'objet statique « Pouce sur le mod, lui-même sur les deux doigts 1 et 2 », au sein duquel le mod (m.Wh) s'est vu appliquer la méthode qui lui est associée (turn) : c'est donc un objet dynamique. On remarque que relier des objets statiques (T, 1V2) à un objet dynamique fait de l'ensemble de l'objet décrit un objet dynamique.

On a donc le premier objet suivant : Pouce sur le mod, lui-même sur les deux doigts, ensuite de quoi le mod tournera de 2Pi (tour complet), dans le sens contraire des aiguilles d'une montre, et la rotation sera plane.

Le deuxième objet est un objet statique qui décrit la position finale du mod (il arrive donc après le premier objet) : pouce sur le mod, lui-même sur 1.

La séquence de ces deux objets constitue le thumbaround. J'aurais pu être plus général en définissant le ThumbAround comme prenant en entrée des objets de type doigt, pour définir d'un coup les TA autours de n'importe quel doigt, mais s'aurait été encore moins lisible.

Quel est l'intérêt de tout cela ?

Eh bien, nous disposons maintenant de deux objets : ThumbAround_T2_T1, défini ci-dessus, et Twisted_Sonic, défini dans mon post précédent sur le topic de Skatox
Je vais donc créer un nouveau package, noté t pour tricks. Dedans, je mets simplement ces deux objets (qui sont en fait des classes, mais je ne fais plus la distinction).
J'ajouterai encore le symbole # pour importer le contenu d'un package donné.

Maintenant, prenons un cas concret :

Nous sommes en 2020. Le langage de programmation que j'ai défini porte désormais un nom, disons PSLanguage, et est interprétable par le logiciel PSReader, conçu par la richissime société Lindor&co.
Un nouveau spinneur, disons Georges, qui a bien sûr, comme tous les spinneurs, notre logiciel, contacte notre service d'aide : « bonjour, j'ai entendu parler d'un antique trick, le twisted acme revolution, mais je ne sais pas à quoi il correspond ! ».
Georges est débutant et n'a jamais appris à breakdowner quoi que ce soit. Qu'à cela ne tienne, notre service d'aide lui envoie le programme suivant :

```
“#t
Twisted_Acme_Revolution = {Twisted_Sonic ; ThumbAround}”
```

Bien sûr, Georges ne comprend rien au programme… mais il s'en fiche : il le copie/colle dans son beau logiciel PSReader. Ce dernier lit le programme :

#t : il importe le package t, qui contient des milliers de tricks (l'équipe de maintien le met constamment à jour avec tout ce qui se fait de nouveau). Parmi eux se trouvent les tricks Twisted_Sonic et ThumbAround.
Il créé ensuite l'objet Twisted_Acme_Revolution, ce dernier étant défini comme une séquence de deux objets que notre logiciel connaît déjà.

Enfin, Georges clique sur « Demonstration » : une vidéo s'affiche alors à l'écran, sur laquelle une main virtuelle exécute un twisted sonic, puis un thumbaround. L'enchaînement est saccadé, mais Georges n'aura aucune difficulté à reproduire les deux mouvements et comprendra très vite comment les relier en un seul enchaînement fluide. Il ajoute donc avec bonheur un nouvel enchaînement à son panel de tricks.

C'est présenté avec humour, mais l'idée est là : outre l'exercice intellectuel, définir un tel langage rend tout à fait possible la création d'un logiciel permettant de visualiser un trick ou un enchaînement donné à partir d'un code.
N'importe qui serait libre d'ajouter de nouveaux objets aux différents package : en plus du package t, on pourrait

créer un package c, qui contiendrait un très grand nombre de combos recensés et dument codés et qui serait mis à jour à chaque « grand » combo sorti. N'importe qui pourrait ensuite importer ce package c sur son logiciel PSReader et visualiser les combos qu'il souhaite.

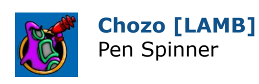
Bref, les applications ne sont pas complètement vides de sens.

Extensions :

Ce que j'ai défini est simplement une base. De très nombreuses choses pourraient être rajoutées : un système plus adapté au XpXh, des objets de type wrist, arm & cie, de nouvelles méthodes (jump, etc)…

Et en effet, avec cette méthode, la notion de trick n'a plus de sens en soit. On peut définir n'importe quel enchaînement et le stocker dans un package donné. L'existences de doublons (plusieurs noms pour des tricks trop proches ou des enchaînements qui ne sont pas des tricks) ne pose aucun problème tant qu'ils sont chacun codés et stockés : le logiciel pourrait alors les lire de toute façon.
Le système que j'ai conçu ne remet en rien en cause le système classique de breakdownage, qui est adapté à une utilisation « manuelle », c'est-à-dire sans passer par un logiciel consacré. Mon système est complètement différent et ne s'utilise pas de la même façon.

Je vous aime tous, sauf un.


Chozo [LAMB]
Pen Spinner



Re: Un langage informatique descriptif des combos

lun. 12 déc. 2011 18:24

C'est très, très stylé comme perspective.

> **YamYam a écrit :**
> Si on est pas d'accord avec quelque chose c'est forcément qu'on trolle ?

Afficher


Yaemgo
Pen Spinner



Re: Un langage informatique descriptif des combos

lun. 12 déc. 2011 18:29

Bonjour.
Tu es fou.
Au revoir.

Sinon, ça me paraît excessivement compliqué par rapport à l'utilité du biniou. Evidemment, si des gens sont motivés pour faire ça, j'en serais franchement ravi ; ton idée Lindor est excellente et on sent un gros travail de réflexion derrière, cependant c'est beaucoup de boulot.
(+j'ai ri du twisted acem revoluïcheune)

And I was like, "Was ist das ?"
Image

> **Lanfear a écrit :**

> Et là je retrouve Yam en train d'embrasser Sheepside



mEat*
est innocent !

Re: Un langage informatique descriptif des combos

lun. 12 déc. 2011 19:09

Très intéressant. Ça rejoint ce que je pensais sur les langages informatiques, ils sont partout. Tu codifies la vie avec ça.



Seifer
Pen Spinner



Re: Un langage informatique descriptif des combos

lun. 12 déc. 2011 19:31

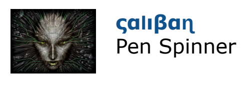
Tu as totalement raison sur ce qui t'as amené à la reflexion (la façon de faire les figure n'est pas dit) mais ce type de notation est (ou du moins semble être) trop compliqué pour les non initiés.

Cependant good job pour ce topic qui a l'air murement réfléchit

---

topic vidéo
http://thefpsb.penspinning.fr/videos-se ... 12737.html
Solo Video special Holiday part 2
http://thefpsb.penspinning.fr/solo/seif ... 13495.html
re up de FPSB Anthology sur ma chaine.

> **Palmito a écrit :**
> t'a afficher devant tout le monde que j'etais un trans :(



çalıβaŋ
Pen Spinner



Re: Un langage informatique descriptif des combos

lun. 12 déc. 2011 20:56

le principe est génial . faire un programme ce serait bueno mais IMO pour apprendre vaut mieux une bonne vielle vidéo ou un commentaire écrit qui me semble tout a fait compréhensible. (la preuve des centaines de gens ont appris comme cela)
après c'est vrai qu'au niveau du travail c'est juste énorme quoi.
mais je trouve que c'est différentes idées de notations ne seraient utile que pour des tricks compliqué ou jamais breaké
je ne voie strictement pas l'intérêt de breaké de cette manière des combos avec des tricks basiques, avec le système des hybrides et du "+" (le truc de freeman) ainsi que la notation de fel tout est breakable.
donc ca peut être intéressant mais pas utile IMO.

VicGotGame, s777, fire@fox, ivabra, fel2fram, gibki, kaede*,sAku, Near, gisele8 > ALL

<3 Crew Cuts <3

Afficher

Afficher



Afficher

Afficher



mEat*
est innocent !



Re: Un langage informatique descriptif des combos

lun. 12 déc. 2011 21:17

Ouais, avec le logiciel t'as une vidéo. Vu que la main serait modélisée, tu pourrais lier des tricks pour faire le combo voulu je pense, mais bon c'est de la théorie. De plus, c'est vrai qu'une vidéo "normale" serait mieux.

Pour finir, on sait tous qu'il est plus simple de breaker normalement.



Taeko
Paveur

Re: Un langage informatique descriptif des combos

lun. 12 déc. 2011 23:59

Très rigolo comme idée, mais malgré 3 ans de Si j'ai toujours du mal à comprendre ne serait ce qu'un peu de prog basique ._.

C'est un beau travail, il faudrait une équipe de fous pour le réaliser.

Afficher

http://s1089.photobucket.com/albums/i354/Taekokonut/

Force kaki gras dégueulasse 



Leftfinger
Doigt parti

Re: Un langage informatique descriptif des combos

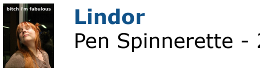
mar. 13 déc. 2011 00:09

edit: Non en fait mon post constructif ne l'est pas parce que je suis pas un crack en informatique alors que c'est ma matière coef 12.

Leftfinger Session 3 !

Afficher

Image



Lindor
Pen Spinnerette - 2 avertissements

Re: Un langage informatique descriptif des combos

mar. 13 déc. 2011 00:23

Dans l'optique ou un tel projet serait accompli, il ne serait pas nécessaire de comprendre ce langage, bien évidemment complexe, pour en bénéficier : on pourrait utiliser le logiciel pour modéliser des enchaînements de tricks & combos présents dans les packages fournis. Savoir le programmer n'est utile que pour mettre à jour la base de donnée avec de nouveaux enchaînements à la mod, de nouvelles idées, etc. C'est une "idée de logiciel" concue pour évoluer - en tout cas, ce ne serait utile qu'en évoluant.

L'utilité par rapport au break n'apparait évidemment que pour les combos compliqués. Un break tordu et imprécis pourrait être remplacé dans un premier temps par la notation formelle que j'introduis, puis, dans un second temps, le nom associé au combo se suffirait en lui même puisqu'il suffirait de l'entrer dans le logiciel pour générer le combo associé.

Mais, et je le redis, c'est surtout l'exercice intellectuel et théorique qui me motive. Le fait de démontrer qu'un tel langage peut tout à fait exister. Comme tout matheux, je me fiche bien de le réaliser : il me suffit de montrer qu'il est réalisable =)

Quant à la somme de travail que cela représente : l'intégralité du post présenté ci-dessus a été conçue durant un cours de japonais d'1h30, pendant lequel j'écoutais quand même (vaguement) ce que disait la prof. J'ai ajouté d'autres détails qui me venaient à l'esprit en rédigeant le post. Ce n'est pas excessivement long à faire, pas du tout. Le travail ne se situe qu'au niveau de la compréhension, je pense. La conception du logiciel en lui-même prendrait plus de temps, mais c'est loin d'être aberrant.

Par contre, concevoir le logiciel ne m'intéresse pas. Pour le moment, en tout cas =)

Je vous aime tous, sauf un.


Lindor
Pen Spinnerette - 2 avertissements





Re: Un langage informatique descriptif des combos

mar. 13 déc. 2011 17:53

Hop, petit ajout : le post n'était pas complet, j'ajoute donc ici un petit paragraphe qui introduit une notion tout en fournissant quelques explications.

On peut utiliser le langage que j'ai défini de manière un peu plus générale que ce que j'ai montré.

Exemple :

T/Side1.*turn*(Pi, CCW, 2D).1 est un trick très simple ; je laisse ceux qui sont motivé tenter de vérifié s'ils sont capable de deviner ce que c'est. Petit indice : même si le mod dans son entier n'est pas mentionné dans la description, une telle notation porte toujours sur le mod entier, quoi que décrivant spécifiquement ce qui se passe pour les parties du mod/de la main qui sont "concernées" par le trick.

Je fournis la réponse en spoiler :

Afficher

Bon, ce n'est pas exactement là où je souhaitais en venir. L'idée était plutôt de dire que j'aurais aussi pu écrire ça :

Finger(a)/Side1.*turn*(Pi, CCW, 2D).Finger(b, b=/=a)

Ce trick est le même que précédemment... à une différence près : les doigts T et 1 sont remplacés par les variables a et b, avec b différent de a.

Cette notation donne donc le trick général, indépendamment du slot. Au moment où on demande au "logiciel" de créer la vidéo associer, le logiciel demande d'affecter des valeurs aux variables.

Qu'est ce que cela implique ? ça implique que le logiciel ne se cantonne pas à lister des tricks en tas, sans faire la moindre distinction : on peut créer des programmes qui correspondent à des schémas de tricks. Ces schémas sont des **objets variables** qui dépendent d'un certain nombre de variable, et on créé une liste d'objets, statiques ou dynamiques, que je nommerai **sous-objets** en affectant une valeur à ces variables.

Bien sûr, on remarque immédiatement que cela peut poser problème si on peut affecter n'importe quelles "valeurs" à nos objets : on pourrait créer ainsi des schémas d'objets (ou objets variables) dont certains sous-objets n'auraient aucun sens.

Cela contraint à définir initialement des ensembles de valeurs, dans lesquels nos variables peuvent prendre leur valeur.
Ces ensembles doivent être ajoutés à la liste de base du "matériel" fourni par le logiciel.

Exemple : je noterai les ensembles entre crochets, et j'appellerai par exemple [F] l'ensemble qui contient les objets 1, 2, 3, 4, T, (F pour finger), et [H] l'ensemble contenant 1,2,3,4,T,P et B (H pour hand) (au cas où, je précise : B c'est BackHand). On peut, si c'est nécessaire, définir d'autres ensembles.
Lorsqu'on veut utiliser une variable, il faut préalablement la définir comme appartenant à un ensemble donné. On peut par exemple utiliser la notation "[ensemble] variable"

Je réécris donc de façon plus réaliste mon "schéma d'objet" précédent :

Schéma_de_Trick-1 =
{ [F] a ;
[F]\[a] b ;
Finger(a)/Side1.*turn*(Pi, CCW, 2D).Finger(b) ; }

Nb : j'ai aussi utilisé les notations mathématiques classiques. [F]\[a] signifie "l'ensemble F privé de a"

Cette fois, j'ai défini formellement un objet variable qui me donne une liste d'objets bien définie, et tous corrects.

Beaucoup plus vaste encore :

Schema_Around =
{ [H]V[Palm] a, b ;
[P(F)]\[a] c ;
a/m.Wh.turn(2Pi, ø, 2D).o(b)/c ; }

Ce nouvel objet est assez intéressant. Il s'agit de l'objet qui introduit le concept d'around.
Encore une fois, j'ai utilisé une notation mathématique sans la définir précédemment, je le fais donc : P(F) signifie, en mathématiques, l'ensemble des parties de F, c'est à dire l'ensemble dont les éléments sont "un ensemble d'objets de F". Par exemple, 1 est dans F, et 1V2 (c'est à dire "1 et 2") n'est pas dans F (qui ne contient que 1,2,3,4 et T), mais est dans P(F).
J'ai aussi utilisé ø :j'aurais pu laisser un blanc à la place, mais bon. J'avais déjà expliqué que préciser les attributs était facultatif. Ici, j'ai donc choisi de ne PAS donner l'attribut correspondant au sens de rotation (CW ou CCW).

Que décrit donc cet objet ?

Il décrit ceci :

Un doigt (ou la paume) est situé sur le mod, lui même situé sur un autre doigt (différent du premier). Bref, le mod est dans un slot quelconque, entre deux doigts ou entre un doigt et la paume.

Ensuite, le mod effectue une rotation d'un tour complet, dans un plan, autour d'un ensemble de doigts (ça peut être de un à 5 doigts, donc). Le sens de la rotation n'est pas précisé.

Ceci est le schéma général d'un around. On peut le contester, en définir un plus vaste, un plus précis, un différent, mais on s'en fiche : l'idée est que ce langage permet non seulement de noter tous les tricks, mais aussi toutes les familles de tricks qu'on peut concevoir. Si l'on conçoit un schéma d'objet dont on peut tirer deux tricks, alors d'une certaine façon, ces tricks appartiennent à la même famille.

Voilà voilà, j'aurais encore pas mal de trucs à dire sur le sujet, mais j'ai un peu la flemme. Je l'écris simplement pour ne pas l'oublier, même si à force de rajouter des trucs, mon travail devient un peu brouillon - il faudrait que liste clairement tout ce qui "constitue" le langage.

Je vous aime tous, sauf un.



Zeta
Pen Spinner

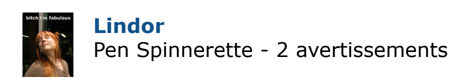
Re: Un langage informatique descriptif des combos

mar. 13 déc. 2011 23:43

Je respecte ton travail Lindor mais la, il te faudra une equipe de fous-geek-lover pour realiser ce programme... (je veux bien etre le bêta tester 🙂)

Afficher

Left's tutos fan



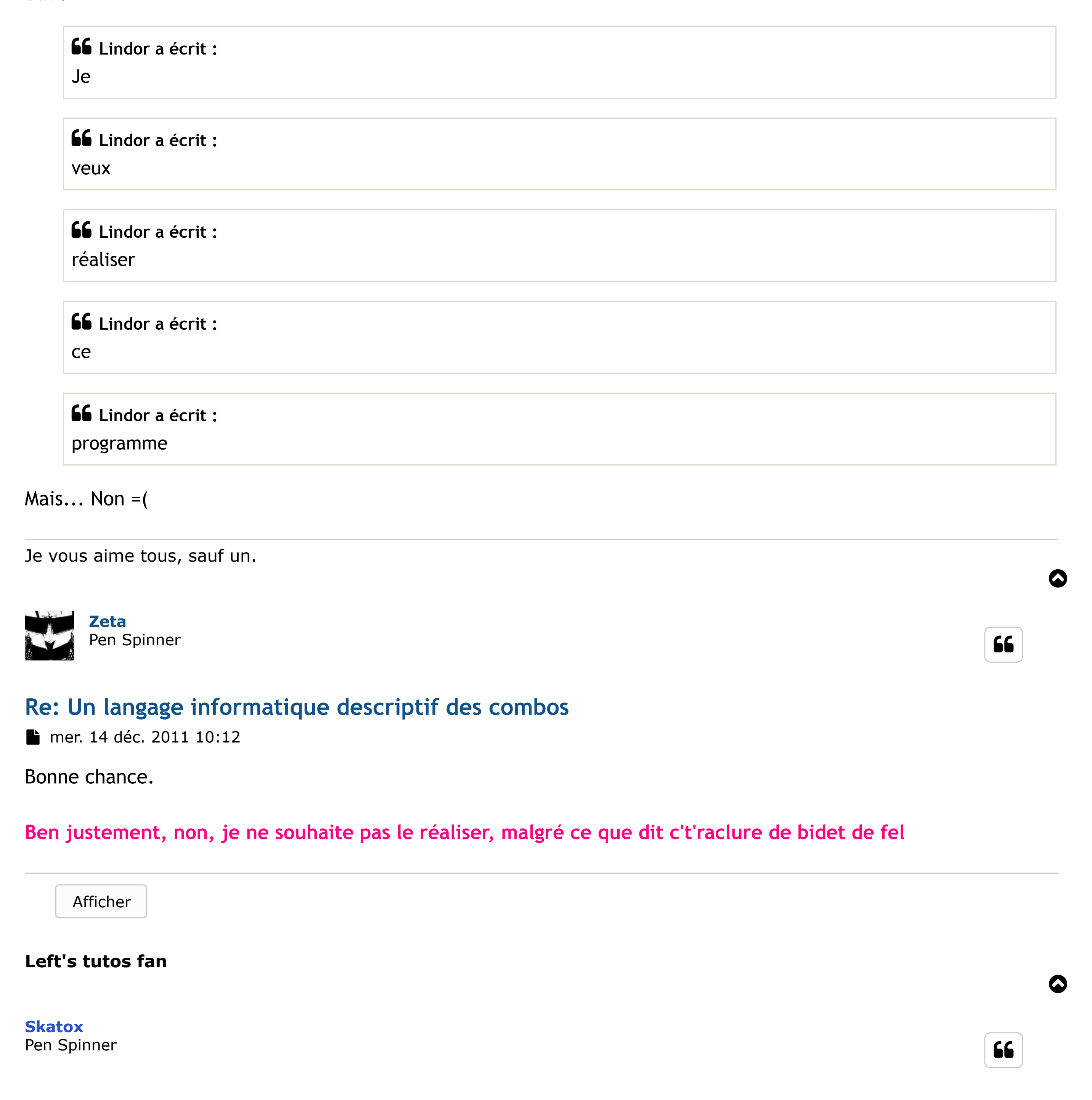
Lindor
Pen Spinnerette - 2 avertissements

Re: Un langage informatique descriptif des combos

mar. 13 déc. 2011 23:53

J'ai dit que je voulais réaliser ce programme ?

:p



oui :

> **Lindor a écrit :**
> Je

> **Lindor a écrit :**
> veux

> **Lindor a écrit :**
> réaliser

> **Lindor a écrit :**
> ce

> **Lindor a écrit :**
> programme

Mais... Non =(

Je vous aime tous, sauf un.

Zeta
Pen Spinner

Re: Un langage informatique descriptif des combos

mer. 14 déc. 2011 10:12

Bonne chance.

Ben justement, non, je ne souhaite pas le réaliser, malgré ce que dit c't'raclure de bidet de fel

---

Afficher

Left's tutos fan

Skatox
Pen Spinner

Re: Un langage informatique descriptif des combos

sam. 17 déc. 2011 04:01

Je ne suis pas persuadé que la programmation orientée objet soit la plus adaptée au problème du penspinning. notamment parce qu'on travaille avec un nombre d'objets finis (à part l'infinité de tricks ou combos qu'on peut décrire), et que la syntaxe objet classique (objet.methode(args)) n'est pas forcément la plus attrayante. Par contre, la quantité d'infos que tu arrives à caser en quelques dizaines de caractères grâce à cette notation, tout en restant un minimum verbose, est assez intéressante !

Comme tu le dis, nos objectifs diffèrent. En particulier, ma notation n'était pas faite pour être interprétée par une machine, mais bien par un oeil humain et a priori non-programmeur. Et le fond est différent aussi car l'approche et les fondations descriptives ne sont pas les mêmes.

Une critique à la volée, il faudrait que j'examine de plus près :

à l'opposé de ma notation, la tienne prend le temps comme une suite d'actions à effectuer. Normal, tu as un langage de type impératif (si si). Mais du coup, comment fais-tu pour vérifier si un programme est "correct", au sens de "réalisable" ? En modélisant le penspinning ainsi, il est très difficile de déterminer quels sont tes variants, invariants... L'un des points que j'appréciais dans ma notation, c'était que je passais par des états fixes définis.

mon style :

état1 transition12 état2 transition23 (etc ...)

les transitions n'étant pas déterminées par les états de départ et d'arrivée, je dois les représenter.

ton système ne décrit que les transitions, sans représenter les états. Peut-être que c'est fiable, mais peux-tu le garantir ? Peux-tu être sûr de ce que tu décris lorsque tu ne décris que les différentes actions à effectuer, sans introduire de "checkpoints" ? J'aimerais croire que tu es dans ton droit, mais j'ai du mal 🙂

